

# Osiguravanje, testiranje i balansiranje opterećenja aplikacije

Pošto smo do sada uvideli šta je sve potrebno da bi se kreirala jedna Django aplikacija i kako se ona postavlja na web, odnosno kako se čini dostupnom svima, potrebno je razmisliti i o aspektu sigurnosti. Postoji dosta metoda i načina da se sajt dodatno osigura i konstantno se otkrivaju nove metode za to. U ovoj nastavnoj jedinici ćemo navesti neke od značajnijih metoda koje se tiču sigurnosti sajta.

## Verzija

Verzija Djanga igra bitnu ulogu u sigurnosti i bitno je koristiti najnoviju stabilnu verziju. Na primer, Django 1.11 verzija (u ovom kursu koristimo Django 3.0.8) bila je poslednja verzija koja je podržavala Python 2.x. Takođe, izbor verzije Djanga direktno uslovljava u kom procentu i kojim napadima će naša aplikacija biti podložna.

## Autentikacija

Iako u našoj Django aplikaciji nismo imali primer autentikacije korisnika, to ne znači da se ta funkcionalnost ne koristi često. Postoje *brute force* napadi, koji mogu poslati više stotina zahteva našem serveru u nameri da pogode ispravan par korisničkog imena i šifre. Kako do toga ne bi došlo, bitno je pravilno podesiti Django aplikaciju da podržava tek određeni broj mogućih pogrešnih pokušaja autentikacije.

## Izvorni kod sajta

Iako je kod na zaštićenom serveru, ne znači da nije podložan napadima, najčešće njegovim neplanskim izvršavanjem. Jedan od najlakših načina da se ovaj problem reši jeste da se čitav kod pomeri „dublje” na serveru, tj. da ne bude u početnom, korenskom direktorijumu servera. Takođe, ako se kod sajta postavlja i na GitHub, GitLab i slične platforme, poželjno je koristiti privatni repozitorijum.

## Korišćenje headera

Radi veće sigurnosti komunikacije klijenta sa serverom, dobra je praksa koristiti referer request polje u zaglavlju. Ono nam omogućava da klijent pošalje informaciju sa kog linka nam šalje zahtev. U slučaju da link sa kojeg dolazi zahtev ne pripada serveru poznatim linkovima, treba obavestiti korisnika.

## Otpremanje (upload) fajlova

U web aplikacijama koje dozvoljavaju korisniku da pošalje fajl treba biti posebno obazriv. Naime, mogući su napadi koji se izvršavaju upravo iz tog fajla, bez obzira na to što naša aplikacija može očekivati samo određeni tip fajla. Primera radi, nekada je bio čest problem da server očekuje sliku tipa .jpg, a da je napadač u tom fajlu, iako je zaglavlje tog fajla podesio da izgleda kao slika, pripremio maliciozni kod koji će se izvršiti na serveru. Kako bi se ovi i drugi problemi rešili, važno je ograničiti koji tip fajla se sme slati i veličinu fajla, validirati sam fajl, kao i ograničiti mogućnost egzekucije koda iz tih statičnih fajlova.

Što se tiče samog Django kao radnog okvira, on je vrlo dobro opremljen kad je reč o sigurnosnim alatima. Otporan je na dosta napada, ali sva ta sigurnost ne dolazi podrazumevana nakon instalacije i kreiranja projekta, već željene sigurnosne parametre moramo sami podesiti.

## Check komanda

Django poseduje sopstvenu komandu za analizu sigurnosti projekata i aplikacija koja se zove check. Za pokretanje te analize, u komandnoj liniji koristimo komandu: `manage.py check --deploy`. Ako bismo je pokrenuli na našem projektu, dobili bismo sledeći ispis:

### Ispis 'manage.py check --deploy' komande:

```
System check identified some issues:
```

```
WARNINGS:
```

```
?: (security.W004) You have not set a value for the SECURE_HSTS_SECONDS
setting. If your entire site is served only over SSL, you may want to
consider setting a value and enabling HTTP Strict Transport Security.
Be sure to read the documentation first; enabling HSTS carelessly can
cause serious, irreversible problems.
```

```
?: (security.W008) Your SECURE_SSL_REDIRECT setting is not set to True.
Unless your site should be available over both SSL and non-SSL
connections, you may want to either set this setting True or configure
a load balancer or reverse-proxy server to redirect all connections to
HTTPS.
```

```
?: (security.W012) SESSION_COOKIE_SECURE is not set to True. Using a
secure-only session cookie makes it more difficult for network traffic
sniffers to hijack user sessions.
```

```
?: (security.W016) You have 'django.middleware.csrf.CsrfViewMiddleware'
in your MIDDLEWARE, but you have not set CSRF_COOKIE_SECURE to True.
Using a secure-only CSRF cookie makes it more difficult for network
traffic sniffers to steal the CSRF token.
```

```
?: (security.W018) You should not have DEBUG set to True in deployment.
```

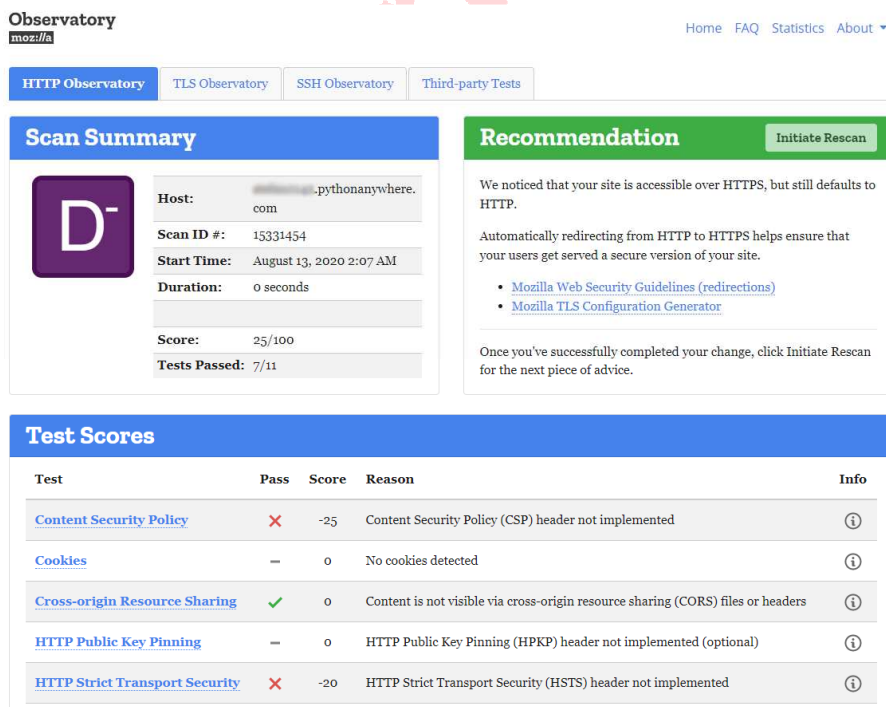
```
?: (security.W022) You have not set the SECURE_REFERRER_POLICY setting.
Without this, your site will not send a Referrer-Policy header. You
should consider enabling this header to protect user privacy.
```

```
System check identified 6 issues (0 silenced).
```

U ovom ispisu vidimo nekoliko upozorenja:

- W004 – upozorenje se odnosi na polje SECURE\_HSTS\_SECONDS settings.py fajla i deo je [deo HTTP Strict Transport Security](#) polise, po kojoj sajt informiše pregledač da se ne povezuje na server za određeni broj sekundi (definisan poljem SECURE\_HSTS\_SECONDS) ako server ne dostavlja pravilno formirane [HTTPS](#) resurse i odgovore;
- W008 – upozorenje koje se opet odnosi na HTTPS i koje kaže da, ako želimo da naš sajt koristi HTTPS umesto HTTP-a, postavimo SECURE\_SSL\_REDIRECT polje na True (preusmeravanje svog HTTP saobraćaja na HTTPS);
- W012 – upozorenje koje se odnosi na kolačiće u zahtevu koje kaže: ako se polje SESSION\_COOKIE\_SECURE u settings.py fajlu postavi na True, klijent kolačiće mora poslati isključivo preko enkriptovane konekcije, ali ne menja samu vrednost kolačića;
- W016 – upozorenje slično prethodnom, ali se odnosi na CSRF kolačić;
- W018 – poruka koja nas upozorava da nam je lokalni server još uvek u razvojnom (development) modu i da je potrebno prebaciti ga u produkcijski mod (DEBUG = False) kada želimo da našu Django aplikaciju postavimo na server;
- W022 – upozorenje koje se odnosi na polje SECURE\_REFERRER\_POLICY u settings.py fajlu, koje je deo polise Referrer-Policy, koja se odnosi na to koliko će se informacija o tome sa kojeg linka dolazi klijent koji šalje trenutni zahtev poslati u našu aplikaciju.

Ako imamo već objavljenu aplikaciju, možemo iskoristiti Observatory sajt kompanije Mozilla, koji će nam skenirati sajt i obavestiti nas o potencijalnim problemima i predloženim rešenjima. Potrebno je otvoriti [sajt](#) i u polju ukucati ime domena aplikacije sa PythonAnywhere platforme. Kliknuti na *Scan Me*. Na strani koja se pojavi dobićemo detaljan izveštaj o sigurnosti, kao na slici:



**Observatory**  
mozilla

Home FAQ Statistics About ▾

HTTP Observatory TLS Observatory SSH Observatory Third-party Tests

### Scan Summary

**Host:** .pythonanywhere.com  
**Scan ID #:** 15331454  
**Start Time:** August 13, 2020 2:07 AM  
**Duration:** 0 seconds  
**Score:** 25/100  
**Tests Passed:** 7/11

### Recommendation

[Initiate Rescan](#)

We noticed that your site is accessible over HTTPS, but still defaults to HTTP.

Automatically redirecting from HTTP to HTTPS helps ensure that your users get served a secure version of your site.

- [Mozilla Web Security Guidelines \(redirections\)](#)
- [Mozilla TLS Configuration Generator](#)

Once you've successfully completed your change, click [Initiate Rescan](#) for the next piece of advice.

### Test Scores

Test	Pass	Score	Reason	Info
<a href="#">Content Security Policy</a>	✗	-25	Content Security Policy (CSP) header not implemented	<a href="#">i</a>
<a href="#">Cookies</a>	—	0	No cookies detected	<a href="#">i</a>
<a href="#">Cross-origin Resource Sharing</a>	✓	0	Content is not visible via cross-origin resource sharing (CORS) files or headers	<a href="#">i</a>
<a href="#">HTTP Public Key Pinning</a>	—	0	HTTP Public Key Pinning (HPKP) header not implemented (optional)	<a href="#">i</a>
<a href="#">HTTP Strict Transport Security</a>	✗	-20	HTTP Strict Transport Security (HSTS) header not implemented	<a href="#">i</a>

Slika 17.1. Observatory izveštaj o sigurnosti naše aplikacije

Kao glavnu poruku u delu Recommendation vidimo informaciju o HTTPS-u. Naime, piše da nam server podržava HTTPS konekciju, a da Django aplikacija koristi HTTP. Na osnovu informacija iz dosadašnjeg teksta prepoznamo da je reč o SECURE\_SSL\_REDIRECT polju u settings.py fajlu. Pa tako, ako na stranici PythonAnywhere otvorimo settings.py fajl i dodamo liniju SECURE\_SSL\_REDIRECT = True, snimimo fajl, osvežimo aplikaciju i opet pokrenemo <https://observatory.mozilla.org/> sajt, videćemo bolji rezultat:

**Observatory**  
mozilla

Home FAQ Statistics About ▾

HTTP Observatory TLS Observatory SSH Observatory Third-party Tests

### Scan Summary

**Host:** pythonanywhere.com  
**Scan ID #:** 15329429  
**Start Time:** August 12, 2020 11:20 PM  
**Duration:** 0 seconds  
**Score:** 45/100  
**Tests Passed:** 8/11

### Recommendation

**Initiate Rescan**

Fantastic work using HTTPS! Did you know that you can ensure users never visit your site over HTTP accidentally?

HTTP Strict Transport Security tells web browsers to only access your site over HTTPS in the future, even if the user attempts to visit over HTTP or clicks an [http://](#) link.

- [Mozilla Web Security Guidelines \(HSTS\)](#)
- [MDN on HTTP Strict Transport Security](#)

Once you've successfully completed your change, click Initiate Rescan for the next piece of advice.

### Test Scores

Test	Pass	Score	Reason	Info
<a href="#">Content Security Policy</a>	✗	-25	Content Security Policy (CSP) header not implemented	<a href="#">i</a>
<a href="#">Cookies</a>	—	0	No cookies detected	<a href="#">i</a>
<a href="#">Cross-origin Resource Sharing</a>	✓	0	Content is not visible via cross-origin resource sharing (CORS) files or headers	<a href="#">i</a>
<a href="#">HTTP Public Key Pinning</a>	—	0	HTTP Public Key Pinning (HPKP) header not implemented (optional)	<a href="#">i</a>
<a href="#">HTTP Strict Transport Security</a>	✗	-20	HTTP Strict Transport Security (HSTS) header not implemented	<a href="#">i</a>

Slika 17.2. Observatory izveštaj o sigurnosti naše aplikacije nakon omogućavanja HTTPS konekcije

U novom izveštaju, u sekciji Recommendation vidimo poruku o tome kako smo ispravili ovu grešku, kao i razliku u oceni. Takođe, na istoj stranici možemo detaljno pogledati koje je tačno od sigurnosnih provera naš sajt prošao, a koje nije i prema tome reagovati po potrebi. Važno je napomenuti da, ako iskoristimo SECURE\_SSL\_REDIRECT = True prilikom rada na lokalnom serveru – dobićemo grešku, jer je sam lokalni server (django web development server) podešen na HTTP.

Detaljnije informacije o sigurnosti u Django pronađite u [dokumentaciji](#).

## Testiranje aplikacije

Testiranje se može predstaviti kao jedan deo koda koji proverava da li drugi, željeni deo koda funkcioniše onako kako smo mi to zamislili.

Testiranje možemo podeliti na više vrsta:

- Izolovano testiranje (unit tests) – kod se izoluje od ostatka kodne baze i tako se samo on testira; ovo su najbrži testovi; najčešće se odnosi na testiranje jedne po jedne funkcije ili klase;
- Integracijski testovi (integrations tests) – testovi koji testiraju više različitih delova koda kako bi proverili da li pravilno funkcionišu; prost primer ovog tipa testiranja bi bila provera podataka između zahteva i odgovora na serveru;
- Funkcionalni testovi (functional Tests) – odnose se na testiranje od početka do kraja. Tačnije, emuliraju pravog korisnika i njegovo ponašanje (emuliraju čitav pregledač) na sajtu. Najsporiji su za izvršavanje.

Testiranje je dosta kompleksan posao, ali takođe od izuzetne važnosti za našu aplikaciju, jer od testiranja direktno može zavisiti i stabilnost aplikacije. Takođe, važno je napomenuti da generalni principi testiranja iz ove nastavne jedinice ne važe samo prilikom testiranja Django aplikacije, već se mogu odnositi na bilo koji kod pisan u Pythonu. Pa tako, nevezano za Django, u Pythonu postoji rezervisana ključna reč `assert`, kojom možemo proveriti bilo koje dve naredbe, promenljive itd.:

```
assert "hello" == "hello", "Hello World, error!"
assert "hello" == "world", "Hello World, error!"
```

Ako bismo pokrenuli ovaj kod, prva linija bi se izvršila bez ikakvog ispisa, ali bi se već kod druge linije pojavila `AssertionError` greška. Dakle, povratna vrednost naredbe `assert` će postojati samo u slučaju da uslov koji proveravamo nije tačan. Takođe, pored prvog parametra koji prosleđujemo prilikom korišćenja `assert` ključne reči – uslova koji proveravamo, postoji i drugi, opcioni, koji se odnosi na poruku koja će se ispisati pored `AssertionError`.

## Testiranje u Django

U sledećem primeru ćemo se fokusirati direktno na testiranje naše aplikacije `book_library`. U tom folderu treba naći fajl `tests.py`, koji je po kreiranju aplikacije uvek prazan.

Testiranje se vrši kreiranjem klase koja nasleđuje `TestCase` klasu, koja je deo `django.test` biblioteke. Iz iste biblioteke treba uvesti i klasu `Client`, koja će nam poslužiti za slanje GET i POST zahteva. Pored ova dva importa, u `tests.py` uvesti i funkciju `reverse` pomoću koje ćemo pozivati linkove nad kojima želimo da izvršimo testove. Opciono, ako želimo dozvoljen pristup promenljivama u `views.py` fajlu, možemo uvesti i `from . import views`.

Metode klase koju ćemo kreirati, koja nasleđuje `TestCase` klasu, malo su specifične i podržavaju samo određenu sintaksu. Naime, postoji specifična metoda `setUp(self)`, koja će se izvršiti pre svih ostalih testnih funkcija, pa je tako možemo iskoristiti za postavljanje scenarija za testiranje. U našem slučaju želimo da je iskoristimo samo za inicijalizaciju klijentskog objekta klase `Client()`, pa tako naš dosadašnji kod fajla `tests.py` izgleda ovako:

```
# Create your tests here.
from django.test import TestCase, Client
from django.urls import reverse
from . import views
class ExampleViewTestCase(TestCase):
    def setUp(self):
        self.client = Client()
```

Sada možemo ispisati svoj prvi test. Metode u našoj klasi `ExampleViewTestCase` koje će sadržati logiku za testiranje moraju početi sa `test_` kako bi prevodilac znao da se upravo na testiranje i odnose. Na primer, ako bismo u klasi `ExampleViewTestCase` imali metodu koja ne počinje sa `test_`, ona ne bi bila izvršena. Za početak, želimo da testiramo da li klijent, ako otvori početnu stranicu `books/`, dobija `index.html` šablon, kao i da li je taj zahtev uopšte i uspešan (statusni kod):

```
def test_index_GET(self):
    response = self.client.get(reverse('index_page'))
    self.assertEqual(response.status_code, 200)
    self.assertTemplateUsed(response, 'index.html')
```

Prvo smo pozvali metodu `get` klijentskog objekta kome smo prosledili `reverse()` funkciju sa parametrom `index_page`, koji predstavlja imenovani link iz fajla `urls.py`, liste `urlpatterns`: `path('books/', views.index, name = 'index_page')`. Odgovor koji pristigne iz ovog GET poziva smeštamo u `response` pomoću kojeg proveravamo:

- da li je statusni kod odgovora 200 – dakle, da li je GET zahtev uspešan: `self.assertEqual(response.status_code, 200)`; u ovom slučaju proveru vršimo metodom `assertEqual`, u kojoj obe komponente uslova moraju biti apsolutno jednake kako bi uslov bio tačan;
- da li nam je GET zahtev ka `books/` stranici vratio `index.html`: `self.assertTemplateUsed(response, 'index.html')`; u ovom slučaju proveru vršimo metodom `assertTemplateUsed`, gde proveravamo ispravnost šablonskog fajla dobavljenog nakon GET zahteva.

Za pokretanje ovog testa koristimo sličan poziv kao i za pokretanje servera:

```
python manage.py test book_library -v 2
```

Dakle, pozicioniramo se u komandnoj liniji u direktorijum koji sadrži `manage.py` fajl. Ostali argumenti ove komande su sledeći:

- `test` – komanda test kojom naglašavamo da želimo testiranje;
- `book_library` – naglašavamo da želimo da testiramo upravo našu `book_library` aplikaciju – što znači da će se za testiranje koristiti `'book_library/tests.py'` fajl;
- `-v 2` – parametar `v` je skraćenica od `verbose`; ovim se omogućava opširniji ispis na ekranu prilikom testiranja (prilikom pokretanja testova eksperimentisati sa vrednostima 1, 2, 3 za `v` parametar).



```

C:\Users\... \first_project> python manage.py test book_library -v 2
Creating test database for alias 'default' ('file:memorydb_default?mode=memory&cache=shared')...
Operations to perform:
  Synchronize unmigrated apps: messages, staticfiles
  Apply all migrations: admin, auth, contenttypes, sessions
Synchronizing apps without migrations:
  Creating tables...
  Running deferred SQL...
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying sessions.0001_initial... OK
System check identified no issues (0 silenced).
test_index_GET (book_library.tests.ExampleViewTestCase) ... ok

-----
Ran 1 test in 0.021s

OK
Destroying test database for alias 'default' ('file:memorydb_default?mode=memory&cache=shared')...

```

*Slika 17.3. Prikaz uspešno odrađenog testa*

U ispisu iz komandne linije nam je najbitnija podvučena linija, jer se upravo ona odnosi na uspešnost testa koji se nalazi unutar `def test_index_GET(self)` metode. Ispod toga vidimo *Ran 1 tests in 0.21s OK*, što znači da su svi testovi koji su pokrenuti (jedan test – jedna metoda) bili uspešni.

Za sledeći test želimo da proverimo kako funkcioniše dobavljanje knjige po imenu sa linka po imenu `str_test` definisanog u `urlpatterns` listi kao:

```
path('books/<str:book_name>', views.index, name = 'str_test')
```

I u tom slučaju ćemo poslati GET zahtev, ali ovog puta ćemo proslediti i argument koji će se mapirati na `<str:book_name>` (dakle, taj argument će poslati `book_name`) kako bismo uspešno poslali ime knjige čiju godinu izdanja želimo da dobavimo:

```

def test_get_book_GET(self):
    response = self.client.get(reverse('str_test', args=['Hamlet']))
    self.assertEqual(response.status_code, 200)
    self.assertEqual(response.content, b"<h1>Book Hamlet was
published in 1603 year.</h1>")

```

U ovoj testnoj metodi, pored `assertEqual` metode koja proverava samo statusni kod, imamo i `assertEqual`, koja će proveriti sadržaj odgovora servera. Tačnije, proverice da li HTML kod dobijen u odgovoru odgovara stringu `"<h1>Book Hamlet was published in 1603 year.</h1>"`. U toj liniji smo takođe ovaj string pretvorili u bajt string, jer HTTP server šalje bajt string kao odgovor.

Kako bismo bili sigurni, zbog nas samih, da testovi nisu prosto preskočeni, što se nekad može desiti usled pogrešno konfigurisanog Django projekta, pogrešne komande itd. – možemo namerno u assert metodama postaviti netačan uslov i tako proveriti da li se testovi izvršavaju. Za potrebe takvog scenarija, možemo u metodi `test_get_book_get(self)` promeniti string koji se očekuje iz:

```
self.assertEqual(response.content, b"<h1>Book Hamlet was published in  
1603 year.</h1>")
```

u:

```
self.assertEqual(response.content, b"<h1>Book The Raw Youth was published  
in 1603 year.</h1>")
```

Primer neuspešnog testa prilikom poziva `test_get_book_GET(self)` izgleda ovako:

self.assertEqual(response.content, b"<h1>Book The Raw Youth was published in 1603 year.</h1>")  
AssertionError: b'<h1>Book Hamlet was published in 1603 year.</h1>' != b'<h1>Book The Raw Youth was publ  
ished in 1603 year.</h1>'  
-----  
Ran 2 tests in 0.027s  
FAILED (failures=1)  
Destroying test database for alias 'default' ('file:memorydb\_default?mode=memory&cache=shared')..." data-bbox="214 330 779 746"/>

*Slika 17.4. Prikaz neuspešnog testa*

Iz tog ispisa komandne linije vidimo da nam je prijavljen jedan test koji nije „položen” (FAILED (failures=1)), metodu koja nije prošla, kao i detaljne informacije o grešci. Vratimo proveru HTML odgovora na tačnu tvrdnju koja će nastaviti prilikom dobavljanja knjige Hamlet.



Nakon što smo uspešno testirali navigaciju kroz sajt, vreme je da testiramo i POST metodu, tačnije popunjavanje i slanje forme. Za te potrebe ćemo umesto `get()` metode iskoristiti `post()` metodu nad klijentskim objektom, kojoj prosleđujemo prvo putanju na kojoj se forma nalazi u vidu imenovanog linka iz `urls/urlpatterns` liste (`path('add-book/', views.get_book, name='book_added')`). Drugi parametar metode `post()` je tipa rečnik i predstavlja imena polja iz forme kao ključeve i vrednosti koje želimo da testiramo, koje prosleđujemo toj formi (tačna imena polja forme se mogu naći u `book_library/forms.py` fajlu). Pošto nakon uspešnog podnošenja imena i godine knjige naša aplikacija šalje zahtev za preusmerenjem (`redirect`) sa statusnim kodom 302 na `book-added/` link, želimo i to da proverimo. Tako naša test metoda za POST zahtev izgleda ovako:

```
def test_book_post_POST(self):
    response = self.client.post(reverse('book_added'), {'book_title'
: 'Oliver Twist', 'book_year' : '1838'})
    self.assertEqual(response.status_code, 302)
    self.assertEqual(response.url, '/book-added/')
```

U ovom testnom slučaju želimo u svoju `views.py/books` listu knjiga da ubacimo i knjigu Oliver Twist. Ali pre toga treba proveriti uspešnost preusmerenog zahteva i da li se nakon dodavanja knjige prebacujemo na `/book-added/` stranicu. Ako je test prošao, znači da ne moramo ništa menjati u kodu `views.py` fajla i možemo nastaviti sa poslednjom proverom – da li je knjiga zaista dodata. Ovo se može učiniti na više načina, a jedan od njih je prosto proveravanje da li se reč *Twist* nalazi u listi knjiga na linku `/books/`:

```
def test_if_book_is_present(self):
    response = self.client.get(reverse('index_page'))
    self.assertEqual(response.status_code, 200)
    self.assertIn(b'Twist', response.content)
```

Pošto je reč o GET zahtevu, koristimo klijentsku metodu `get()`, kojoj opet prosleđujemo link imenovan sa `index_page`. Iz tog zahteva proveravamo statusni kod metodom `assertEqual`, kao i uslov: da li se bajt string *Twist* nalazi u tom HTML odgovoru servera, metodom `assertIn`.

### Pitanje

Ako u ispisu izveštaja pokretanja testa vidimo makar jednu liniju sa rečju FAIL, da li je naša aplikacija uspešno prošla testiranje?

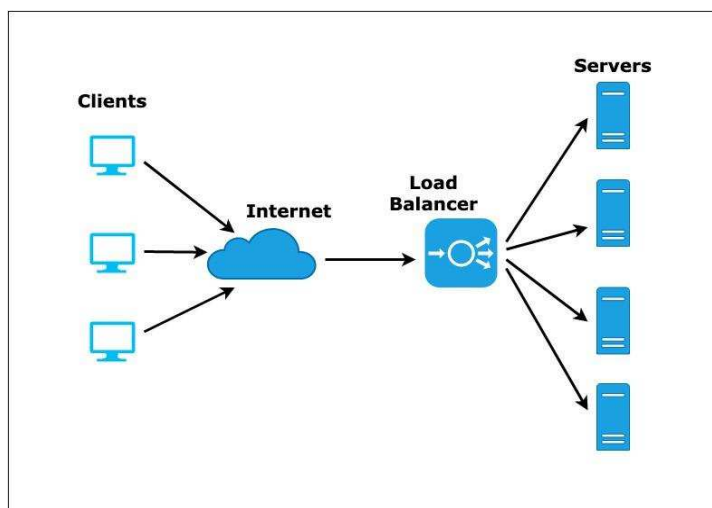
- Da
- **Ne**

### Objašnjenje:

*Tačan odgovor je da naša aplikacija nije uspešno prošla testiranje. Za uspešan prolaz testa aplikacija ne sme imati neuspešno izvršene testove – svi testovi moraju imati status OK.*

## Balansiranje opterećenja

Balansiranje opterećenja (raspoređivanje) predstavlja esencijalnu komponentu u serverskoj arhitekturi. Kao što i samo ime kaže, uloga ovakvih komponenti je da raspodele opterećenje, odnosno, raspodele klijentske zahteve po serverima. Mogu se i čitavi zahtevi zasebno rasporediti, a i resursi potrebni za obavljanje samo jednog zahteva.



*Slika 17.5. Šematski prikaz raspoređivača opterećenja pri klijent-server komunikaciji<sup>1</sup>*

Pošto većina popularnih sajtova danas ima visok saobraćaj gde se klijentski zahtevi konstantno broje u milionima, postoji potreba da se smanji opterećenost servera. Upravo u tim scenarijima se koristi balansiranje.

## Trajnost sesije

Bitan problem koji se javlja prilikom balansiranja predstavlja scenario kako se ophoditi prema informacijama koje se moraju čuvati širom više zahteva u toku korisničke sesije. Ako se ova informacija čuva lokalno na serveru, onda serveri na koje naredni zahtevi odu neće imati informaciju o tome. Tako, ako se balansiranje ne konfiguriše pravilno, može izazvati upravo suprotan efekat i zauzeti više resursa nego što je inače potrebno.

## Način funkcionisanja balansiranja

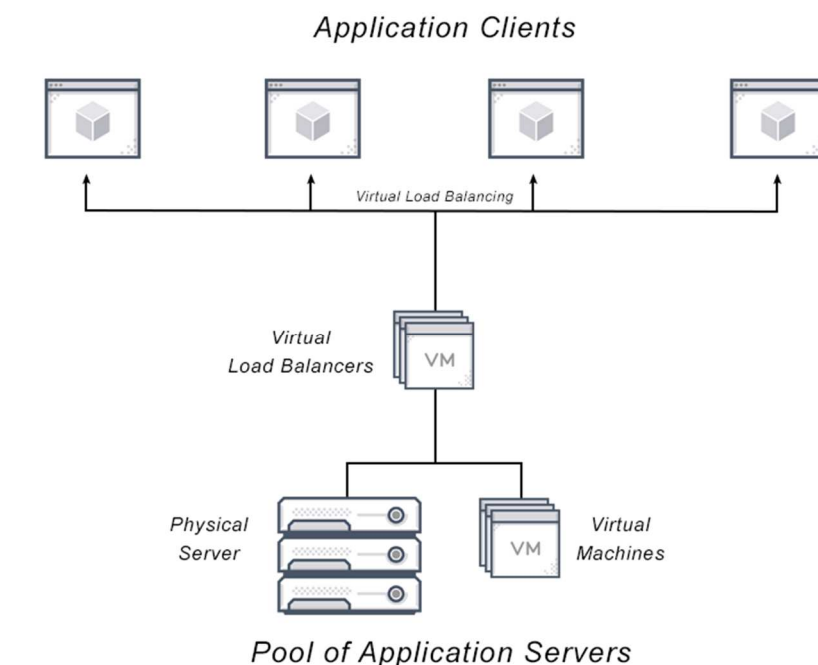
Način na koji takav jedan proces funkcioniše je jednostavan. Balansiranje se uglavnom vrši ili preko softvera ili preko hardvera, a princip se svodi na isto: osluškivanje klijentskih zahteva po određenom portu. Kada zahtev dođe, raspoređivač ga preuzima i po unapred određenoj logici i kriterijumima prosleđuje taj zahtev serveru koji je pod prihvatljivim opterećenjem. Nakon što server obradi zahtev, šalje odgovor raspoređivaču, koji taj odgovor prosleđuje klijentu. Na taj način klijent nije svestan ove komunikacije raspoređivaču – server i ne može znati kako je određena podela web funkcionalnosti širom servera.

<sup>1</sup> <https://codeburst.io/load-balancers-an-analogy-cc64d9430db0>

## Tipovi balansiranja

Postoji više različitih tipova balansiranja:

- elastično balansiranje – automatski proširuje potrebne resurse na osnovu količine saobraćaja; to čini tako što konstantno prati iskorišćenost servera i prema unapred određenoj logici procenjuje i alocira više resursa;
- geografsko balansiranje – odnosi se na distribuiranje saobraća širom različitih servera i data centara kako bi se zahtevi klijenata podelili po geografskoj lokaciji sa koje dolaze;
- virtuelno balansiranje – odnosi se na rad sa virtuelnim mašinama i serverima.



Slika 17.6. Šematski prikaz virtuelnog balansiranja<sup>2</sup>

## Softversko i hardversko balansiranje

Hardversko balansiranje se u praksi najčešće pojavljuje kao rešenje koje kompanija nudi zajedno sa svojim softverom, koje podrazumeva specijalno dizajniran hardver za namene balansiranja. Ova rešenja su obično skupa i preporučuju se za velike sajtove.

Prednost softverskog balansiranja je to što taj softver može raditi na već postojećem hardveru koji posedujemo, pa je tako jeftinije i fleksibilnije, ali zato i manje efektivno kada je reč o sajtovima sa milionima zahteva.

<sup>2</sup> <https://avinetworks.com/glossary/virtual-load-balancer/#:~:text=A%20Virtual%20Load%20Balancer%20provides,appliance%20on%20a%20virtual%20machine>

## Prednosti i mane balansiranja

Postoji dosta prednosti kod implementiranja balansiranja na sajtu. Neke od tih prednosti su:

- manji vremenski period koji će server provesti kao nedostupan, jer ako padne jedan server – drugi će prihvatiti njegove zahteve;
- ostavlja se bolji utisak kod klijenata, jer se sajt brže učitava, pa tako klijent brže dolazi do željenih informacija; u ovom slučaju raspoređivači opterećenja su konfigurisani tako da ili rasporede zahteve po serverima ili korisniku dobave željeni podatak sa njemu geografski najbližeg servera;
- manje opterećenje jednog servera; ovo se odnosi i na prethodnu stavku; pošto je jedan server manje opterećen, klijenti će brže dobijati odgovor, a i manje su šanse da će taj isti server biti pretrpan zahtevima.

Naravno, postoje i mane implementacije balansiranja. Ako to činimo sami – samo će se povećati kompleksnost čitavog servera, a i uloženo vreme. Ako koristimo već napravljena rešenja, bilo softverska bilo hardverska – moramo platiti za te usluge, pa je tako dobro prvo analizirati da li je našem sajtu uopšte i potrebno balansiranje opterećenja ili jednostavno ulaganje u već postojeći server.

## Uloga komponente balansiranja

Pošto smo uvideli neke od prednosti i mana balansiranja, pogledajmo i koje su njegove uloge:

- kao što smo već pomenuli, jedna od primarnih uloga je distribuiranje zahteva širom servera na osnovu predefinisanih algoritama za balansiranje;
- aktivno praćenje „zdravlja“ servera u serverskom klasteru, odnosno praćenje njegovog stanja i zauzetosti resursa;
- olakšava dinamičko dodavanje i sklanjanje liste servera, što poboljšava proširivost.

## Slojevi mreže i deljenje opterećenja

Videli smo da se raspoređivači balansiranja mogu podeliti na softverske i hardverske. Takođe, mogu se podeliti i prema sloju OSI modela u kojem su implementirani:

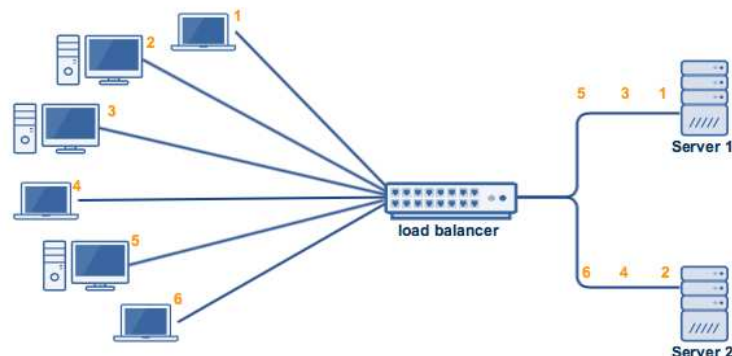
- raspoređivači balansiranja pri transportnom sloju (L4 Load balancers) su raspoređivači koji funkcionišu na četvrtom sloju OSI modela. IP adresa i TCP port su jedine dostupne informacije iz mrežnih paketa četvrtog sloja, pa tako ovi raspoređivači imaju ograničenu funkcionalnost;
- raspoređivači balansiranja pri aplikativnom sloju (L7 Load balancers) su raspoređivači koji funkcionišu na aplikativnom sloju OSI modela. S tim u vezi, imaju pristup mnogo većem setu podataka nego raspoređivači iz četvrtog sloja, što uključuje HTTP zaglavlja, podatke podnesene u formi, sesijski broj itd; ovaj dodatni set informacija čini raspoređivače sedmog nivoa inteligentnijim i rasprostranjenijim.

## Algoritmi balansiranja

Pošto smo se upoznali sa principom rada raspoređivača, pomenućemo neke od najkorišćenijih algoritama za raspoređivanje zahteva/resursa:

### Round robin algoritam

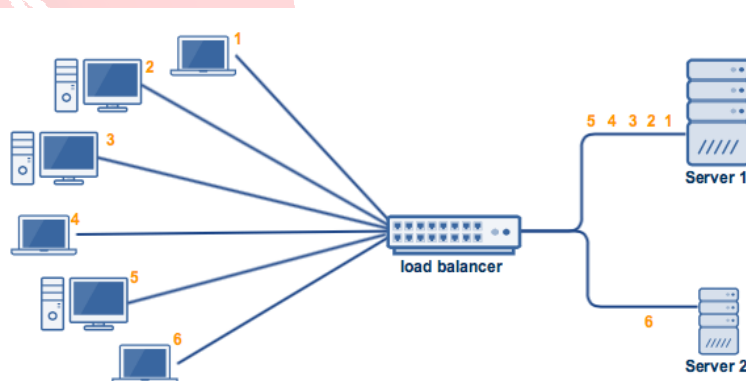
Kada se koristi ovaj algoritam, server raspoređuje zahteve tako što pristigli zahtev pošalje serveru, on ga obradi i pošalje zahtev nazad. Naredni zahtev ide sledećem serveru i tako ukруг u okviru serverskog klastera. U praksi se ovaj metod ređe koristi.



Slika 17.7. Šematski prikaz round robin algoritma<sup>3</sup>

### Selektivni round robin algoritam (weighted round robin)

Funkcioniše na sličan način kao i round robin metoda, ali se svakom od servera dodeljuje broj koji praktično označava koliko taj server može podneti zahteva. Što je taj koeficijent veći, može podneti više zahteva i na osnovu tog broja raspoređivač raspoređuje zahteve.



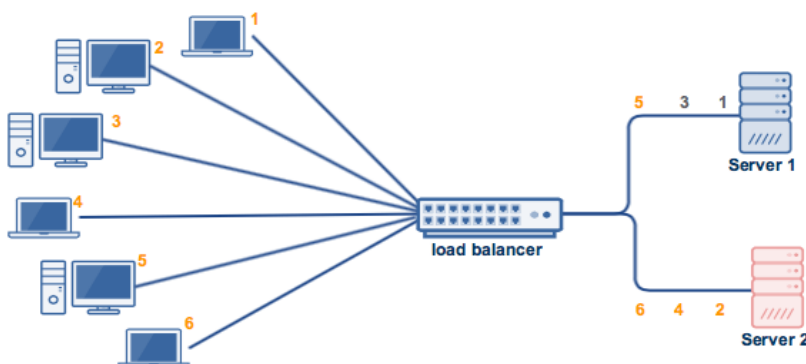
Slika 17.8. Šematski prikaz selektivnog round robin algoritma<sup>4</sup>

<sup>3</sup> <https://www.cybercureme.com/load-balancer-how-does-it-work-with-reconnaissance-phase-during-penetration-testing/>

<sup>4</sup> <https://www.cybercureme.com/load-balancer-how-does-it-work-with-reconnaissance-phase-during-penetration-testing/>

## Najmanje konekcija (least-connection)

Ovo je jednostavna i jako efikasna metoda. Kao što i samo njeno ime kaže, raspoređivač će pronaći server koji ima najmanje otvorenih konekcija i njemu dodeliti zahtev. Na primer, ako server može opslužiti maksimalno deset konekcija odjednom i trenutno ima otvorene dve konekcije, a drugi server može primiti dvadeset konekcija odjednom a trenutno ima četiri otvorene – raspoređivač implementiran ovih algoritmom će odlučiti da oba imaju isti broj konekcija. Zato se kaže da je ovaj algoritam dosta fer u odnosu na round robin.



Slika 17.9. Šematski prikaz algoritma sa najmanje konekcija<sup>5</sup>

## IP heširanje (IP hashing)

Ovaj algoritam se može primeniti u situacijama kada moramo omogućiti da isti korisnici pristupaju uvek istim serverima ili na primer kada treba odrediti geografski najbliži server. Koristiće heš funkciju, koja će na osnovu IP adrese zahteva odrediti kom serveru treba proslediti zahtev.

## Rezime

- Django poseduje sopstvenu komandu za analizu sigurnosti projekata i aplikacija koja se zove check. Za pokretanje te analize koristimo komandu u komandnoj liniji: `manage.py check --deploy`.
- Ako imamo već objavljenu aplikaciju, možemo iskoristiti sajt Observatory kompanije Mozilla, koji će nam skenirati sajt i obavestiti nas o potencijalnim problemima i predloženim rešenjima.
- Testiranje se može predstaviti kao jedan deo koda koji proverava da li drugi, željeni deo koda funkcioniše onako kako smo zamislili.
- Postoji više tipova testova:
  - izolovani testovi;
  - integracijski testovi;
  - funkcionalni testovi.

<sup>5</sup> <https://www.cybercureme.com/load-balancer-how-does-it-work-with-reconnaissance-phase-during-penetration-testing/>



- Testiranje se vrši kreiranjem klase koja nasleđuje TestCase klasu, koja je deo django.test biblioteke.
- Postoji specifična metoda setUp(self), koja će se izvršiti pre svih ostalih testnih funkcija, pa je tako možemo iskoristiti za postavljanje scenarija za testiranje.
- Metode u klasi koje će sadržati logiku za testiranje moraju početi sa test\_ kako bi prevodilac znao da se upravo na testiranje i odnose.
- Uloga raspoređivača opterećenja je da raspodele opterećenje, odnosno da raspodele klijentske zahteve po serverima. Mogu se i čitavi zahtevi zasebno rasporediti, a i resursi potrebni za obavljanje samo jednog zahteva.
- Postoji više različitih tipova balansiranja:
  - elastično;
  - geografsko;
  - virtuelno.
- Među najkorišćenijim algoritmima za raspoređivanja zahteva/resursa su: round robin, selektivni round robin, najmanje konekcija i IP heširanje

